Randomized Graph Drawing with Heavy-Duty Preprocessing

David Harel* and Meir Sardas†
Dept. of Applied Mathematics and Computer Science
The Weizmann Institute of Science, Rehovot, Israel

March 15, 1995

Abstract: We present a graph drawing system for general undirected graphs with straight-line edges. It carries out a rather complex set of preprocessing steps, designed to produce a topologically good, but not necessarily nice-looking layout, which is then subjected to downhill-only version of Davidson and Harel's simulated annealing beautification algorithm. The intermediate layout is planar for planar graphs and attempts to come close to planar for non-planar graphs. The system's results are better and faster than what the annealing approach is able to achieve on its own.

1 Introduction

A large amount of work on the problem of graph layout has been carried out in recent years, resulting in a number of sophisticated and powerful algorithms. An extensive and detailed survey can be found in [BETT93]. Many of the approaches taken are limited to special cases of graphs, such as trees or planar graphs; others concentrate on special kinds of layouts, such as rectilinear grid drawings, or convex drawings.

In this paper, we continue the work of Davidson and Harel [DH89], which addresses the general problem of drawing arbitrary undirected graphs on the

*Email: harel@wisdom.weizmann.ac.il. Part of this author's work was carried out during visits to INRIA in Sophia Antipolis, and to Cornell University in Ithaca, NY, and was partially supported by grants AF #F49620-94-1-0198 (to F. Schneider), NSF #CCR-9223183 (to B. Bloom), and NSF #CDA-9024600 (to K. Birman).

†Email: meir@orbot-instr.co.il. Current address: Orbot Instruments LTD, Yavneh Industrial Zone, P.O.B 601, Yavneh, Israel

plane, with edges drawn as straight-line segments. The goal is to try to achieve as "nice" a drawing as possible.

The work in [DH89] uses simulated annealing to maximize a cost function that reflects the aesthetic quality of the drawing, according to the following criteria: (i) distributing vertices evenly, (ii) making edge lengths uniform, (iii) minimizing edge crossings, and (iv) keeping vertices from coming too close to edges. The system developed in [DH89] performs well on small graphs, but becomes unsatisfactory when applied to graphs of over 30 vertices or so, especially with respect to minimizing edge crossings. Planar graphs that do not result in planar layouts are particularly annoying.

The basic idea in the present work is to use some rather intricate algorithms and heuristics — part known and part new — to first obtain a rough approximation to a drawing, with special emphasis on minimizing edge crossing, but with very little that has to do with aesthetics, and then to submit the result to (a modified version of) the annealing system of [DH89] for beautification according to the other criteria.

Our system employs several phases. Phase A tests for planarity, and is carried out by the algorithm of [BL76, LEC67], using *PQ-trees*. The system then deals somewhat differently with planar and non-planar graphs. The case of planar graphs is simpler, and in it we carry out the following:

Phase A: Planarity testing.

Phase B: Planar embedding.

Phase C: Planar drawing.

Phase D: Randomized beautification.

Phase B uses the PQ-trees-based algorithm presented in [CNAO85] to construct a planar embedding, i.e., an ordered list of the neighbors of each vertex, which, if layed out appropriately in cyclic order around the vertex, leads to a planar drawing.

Phase C then uses the embedding lists produced by the previous phase to actually draw the graph. The output is a planar drawing with (crossing-free) straight-line edges. To carry out this phase we had to design a special drawing algorithm, which is a generalization of the algorithm of [FPP88, CP90].[1] Phase D is the fine-tuning part of the simulated annealing system of [DH89], slightly modified.

For non-planar graphs, the phases are as follows:

[1] This is the most technically involved part of our work, and we have devoted a separate paper to the detailed description and analysis of this algorithm; see [HS93].

Phase A: Planarity testing.

Phase B^-: Extracting planar subgraph.

Phase B: Planar embedding.

Phase B^+: Reinserting removed edges.

Phase C: Planar drawing.

Phase D': Extended randomized beautification.

Phase B^- uses yet another application of the PQ-trees algorithm, described in [Kan92, JTS89], that attempts to find a maximal planar subgraph in the input graph, by eliminating as few edges as possible. The subgraph produced by this phase is then subjected to the planar embedding algorithm of phase B. Following this, phase B^+ reintroduces the eliminated edges, while trying to minimize the number of crossings that arise by doing so. At each crossing point a new vertex is inserted, yielding again a planar graph.[2]

This planar graph is then drawn in phase C and is beautified by phase D' in a manner similar to that of planar graphs. However, we have had to extend the randomized algorithm of [DH89] with new components that try to overcome distortions introduced by phase B^+.

As far as planar graphs are concerned, our system achieves a noticable improvement over the annealing system of [DH89]. In general, all planar graphs are drawn planar. In fact, planar graphs with 50 vertices yield drawings that have a close-to-perfect look. The running time is also significantly improved, as the inherently slow annealing process is not burdened with having to find a solution, but only with "massaging" a topologically suitable layout into a nice-looking one. Phase D, however, is still by far the most time-consuming part of our system, as can be seen in Section 7. For graphs that can be made planar by extracting a small number of edges, the results are still good, and compared with the annealing system ours has the advantage of being stable: In subsequent runs the results are much the same and are all fairly good, while in the annealing system results can vary widely from run to run — some are acceptable, and some are not.

For graphs that are far from planar (i.e., ones that require the elimination of more than 10 edges or so for planarization), improvements are still required,

[2] We should mention the GIOTTO system of Tamassia, Di Battista and Batini [TBB88]. They were interested in drawing diagrams using the *grid standard*, whereby vertices are placed at grid points and edges are rectilinear. While their goals are quite different from ours, there is similarity in the early stages: They have steps similar to our phases A through B^+. However, the algorithms they use for these steps are different from ours.

and the system's results can still be worse than a manually produced drawing, even for medium sized inputs.

Section 2 describes the simulated annealing system of [DH89], which was the starting point for the current work. We next discuss our treatment of planar graphs: Section 3 contains a brief description of the drawing algorithm of phase C for planar graphs, and Section 4 describes some heuristics used to enhance this drawing algorithm and improve its output. Section 5 describes the changes and extensions introduced for the case of non-planar graphs, including the algorithm used to reinsert edges in phase B^+, and the components added to the randomized algorithm in phase D' to minimize damage caused by the reinsertion.

Section 6 discusses some examples of drawings obtained by the system, with the goal of highlighting the improvements over [DH89]. Section 7 summarizes the asymptotic time-complexity of the various parts of the system, and includes a table of performance statistics for example graphs. Finally, Section 8 contains some directions for future work.

2 Randomized beautification

This part in our system is an adaptation of the work of Davidson and Harel [DH89], in which they applied the simulated annealing paradigm to the problem of drawing graphs nicely. We incorporate their system as our final phase, after a topologically acceptable, but non necessarily nice-looking layout has been found in the earlier phases. In this section we briefly describe the system of [DH89], and our adaptation and use of it.

Simulated annealing tries to find a configuration that minimizes a cost function [LA87] carefully designed to capture the "niceness" of a drawing. Minimization is attempted by a process that starts from some initial random drawing, and repeatedly improves it as follows. Given the current candidate drawing σ, a new candidate σ' is generated, that is close to σ, and the following annealing condition is tested:

> Let E and E' be the values of the cost function at σ and σ' respectively;
> **if** $E' < E$ or *random* $< e^{\frac{E-E'}{T}}$, **then** accept σ' as the current candidate.

Here *random* stands for a real number between 0 and 1, selected randomly, and T is the so-called *temperature*, which is cooled down as the process proceeds. This fragment of the algorithm is called an *annealing step*. Generating the next candidate drawing is carried out as follows:

Choose a vertex v_i and an angle θ at random;
Let P be the current position of v_i; Move v_i to a position Q, such that
the line segment PQ is of length r and forms angle θ with the X-axis.

Here, r is the offset radius. It starts from some initial value, and decreases as the process proceeds. The entire process iterates a large number of annealing steps of the kind described above. This number is proportional to the number of vertices in the graph, and, as mentioned, r and T are decreased as the process proceeds.

The cost function developed in [DH89] takes into account several empirical criteria for nice drawings. These are integrated using normalizing factors denoted below by λ_i, that define the relative importance of each criterion in the overall value. Many of the parameters of the algorithm are open for interactive change by the user. This includes control of the normalizing factors, the number of annealing steps to perform, and more. Here is a brief description of the cost function.

The first component tries to spread out the vertices evenly. For each pair of vertices v_i and v_j, the term $\lambda_1/{d_{ij}}^2$ is added to the cost function, where d_{ij} is the Euclidean distance between v_i and v_j in the candidate drawing.

The next component prevents vertices from being positioned too close to the borderline of the drawing space. The following term, for each vertex v_i, takes care of this:

$$\lambda_2\Big(\frac{1}{{r_i}^2}+\frac{1}{{l_i}^2}+\frac{1}{{t_i}^2}+\frac{1}{{b_i}^2}\Big)$$

Here, r_i, l_i, t_i and b_i are the distances between v_i and the four borderlines — right, left, top and bottom.

The next component tries to make the edges short, by adding the term $\lambda_3 {d_e}^2$ to the cost function, for each edge e, where d_e is e's length.

The next component penalizes edge crossings. A fixed value of λ_4 is added to the cost function for each crossing.

The last component tries to keep vertices from coming too close to edges. For each vertex v_k and edge e_l, the term $\lambda_5/{g_{kl}}^2$ is added to the cost function, where g_{kl} denotes the least distance from v_k to any point on e_l. Since the calculation of this component is very time consuming, we have incorporated two variants of the cost function in our system: The *full* one, in which this component appears, and the *simple* one, in which it is omitted.

[DH89] contains examples obtained by the simulated annealing system, and also compares it with the spring-based methods of [Ead84, KK89], and the method of [FR91]. A related approach to aesthetic drawing of straight-line undirected graphs is [Tun92]). While slow in general, due to the inherent time-consuming nature of simulated annealing, the results are very good for small graphs of size up to 20–25 vertices; larger graphs are much harder to handle. Increasing the number of iterations of the system often helps, but this causes a significant increase in the already quite high running time.

The asymptotic running time of the algorithm of [DH89] is $O(n^2 e)$, where $n = |V|$ and $e = |E|$ for input graph $G = (V, E)$. This follows from the fact that updating the cost function can be done with $O(ne)$ per iteration, and the number of iterations is linear in n. The initial value of the cost function for the first drawing must be calculated from scratch, which also requires $O(n^2 e)$ operations.

In our adoption of the annealing system of [DH89] in phase D, we employ the same cost function and the same method for generating new candidates for a drawing. However, experimentation showed that in our context almost all of the moves accepted were those with $E < E'$, and only very few were the *uphill moves*, i.e., those accepted by the annealing condition $random < e^{\frac{E'-E}{T}}$. Consequently, we decided to remove this condition and test only for $E < E'$ in accepting a candidate drawing. This saves significant running time, with almost no influence on the results. We thus call this part of our system the *randomized* phase, rather than the annealing phase. This downhill-only version of the algorithm appears already in the original system of [DH89], as a *fine tuning* stage that is employed to further improve the drawing after the annealing process ends. Thus, interestingly, we use the fine-tuning process for the same task as it is used in [DH89], but here it is preceded not by a lengthy simulated annealing process, but by quite different, more complex yet faster, methods for reaching a rough initial layout of the input graph.

We use both variants of the cost function, and have found that alternating them seems to have the best effect. Start with a number of simple iterations, i.e., ones that do not test vertex–edge proximity, followed by some iterations that use the full cost function, including this time-consuming test; then again a number of simple iterations, followed by some full ones, etc.

3 Planar graphs: The drawing algorithm

In this section we discuss the drawing algorithm for planar graphs, that constitutes phase C of our system. We comment on our choice of approach, and on the need for a significant modification thereof. A detailed description of the resulting algorithm is outside the scope of this paper, and can be found in [Sar93, HS93].

The input to phase C is a planar graph accompanied by the planar embedding constructed in phase B using the PQ-trees algorithm of [CNAO85]. The output is a planar drawing of the graph that complies with the given embedding. By a planar embedding of a graph G we mean an array of lists, one for each vertex, with v's list containing the edges incident to it in circular order around v in a possible planar drawing of G.

Our algorithm is a considerably generalized version of the drawing algorithm of Chrobak and Payne [CP90], which, in turn, is based on an algorithm by de Fraysseix, Pach and Pollack [FPP88]. This algorithm draws a graph with n vertices on a grid of size $(2n-4) \times (n-2)$; vertices are placed on grid points and edges are crossing-free straight lines. It runs in time $O(n)$, and is quite easy to implement.

3.1 Why this algorithm?

This algorithm was chosen from a number of existing algorithms for drawing planar graphs, all of them with linear running times. The first was developed by Chiba, Onoguchi and Nishizeki [CON85], with an eye towards aesthetics. This possibility was rejected for a number of reasons: First, it is rather complicated and hard to implement. Second, it requires high precision calculations, since vertices do not necessarily lie on grid points, while in the algorithm of [CP90] we need calculations of integer values on the order of n only. This has implications to the spread of vertices on the drawing area; when bounded integer values are used, we have a bound on the minimal distance between two vertices. Third, the algorith mof [CON85] does not work well for our purposes. An implementation of it exists in a software package named *GraphEd* [Him89, Him93]. We tried this system on several examples, and the results were unsatisfactory. In many cases, vertices were clustered in a small region of the drawing area, with few vertices spread over distances that were large compared to the clusters. This spreading becomes very problematic in the randomized phase of our system.

Another algorithm we considered for drawing planar graphs is that of Read [Rea87]. This is a recursive algorithm that draws a graph with n vertices from a drawing of a subgraph with $n-1$ vertices. The algorithm requires the input graph to be maximal, and based on this assumption the last vertex can in fact be placed so that no edge crossings emerge. This algorithm is not very complicated, but there are two advantages to the one we chose over Read's. For one, the algorithm of [CP90] yields a guaranteed lower bound on the distance between vertices, while Read's algorithm might result in the kind of clustering of vertices that occurs in the algorithm of [CON85]. The second difficulty is with the maximality requirement. Although it is not hard to first embed a given graph in a maximal graph and then draw it and remove the extra edges, we explain below why this process should be avoided. And while we have managed to overcome this requirement for the algorithm of [CP90] by devising a new version of it, we did not see a simple way to do so for Read's algorithm.

Another algorithm for drawing planar graphs was devised by Schnyder [Sch90], and it also produces the drawing on a grid, with even smaller grid size. In fact, it guarantees a straight line drawing on a grid of size $(n-2)\times(n-2)$, so that it gives a better spread of vertices. However this algorithm also depends on the input graph being maximal, and in a way that doesn't appear to be easily removable.

Jones et. al. [J^+91] compare and test the algorithms of [CON85, CP90, Rea87, Tut63], especially in terms of performance and the spread of their outputs.

3.2 Why a new version?

The original algorithm of [CP90] requires the graph to be *maximal* planar. However, we want the system to work on planar graphs which are not necessarily maximal. The simplest way to achieve this is suggested in [FPP88], namely, *triangulation*: If the graph is not maximal, dummy edges are added as follows. For every vertex v, if u, w are two neighbors of v, adjacent in the circular ordering of v's neighbors but not connected by an edge, add the dummy edge (u, w).

To achieve linear running time for triangulation, we have to be able to check the existence of an edge in constant time. This can be done by using an $n \times n$ adjacency matrix to represent the graph, and a method suggested in [Rea87] to avoid the quadratic time that zeroing the matrix at the initialization stage would take.(However, this requires quadratic space.)

Thus, the method that emerges is to (i) triangulate the graph, (ii) draw the result by the drawing algorithm, and (iii) delete the dummy edges introduced in the first step. The software package *GraphEd* [Him89, Him93] contains an implementation of this algorithm too, and we could test its performance there. For example, Figure 1(a) shows an example of the output (after deleting the added edges). When this drawing was submitted as input to the randomized process of phase D, Figure 1(b) was obtained, which has a major deficiency: Its external face is drawn concave, in a way too twisted for the randomized phase to overcome. Other examples show similar problems, which are the result of the idiosyncrasies of the triangulation step, whose dummy edges often ruin the structure of the graph, yielding unsatisfactory results.

To overcome this difficulty, we have developed a variation of the algorithm of [CP90], which does not require a triangulated graph, but works directly on the original input graph. Our algorithm constructs the graph in steps, in such a way that a vertex v appears in G_k, the graph constructed in step k, only if at least one of its neighbors appears in G_{k-1}. This avoids the situation of vertex

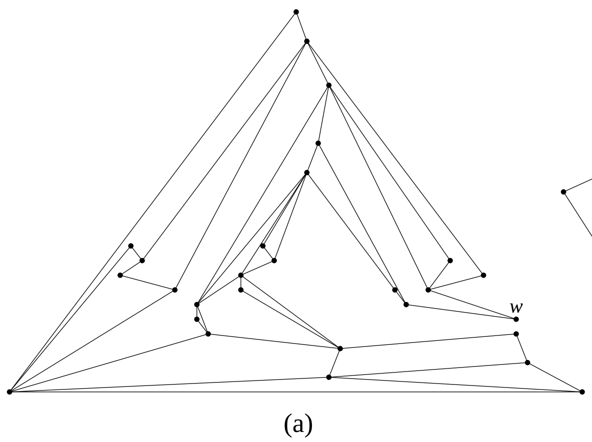


Figure 1: The problematics of naively applying the drawing algorithm to a non-triangulated graph.

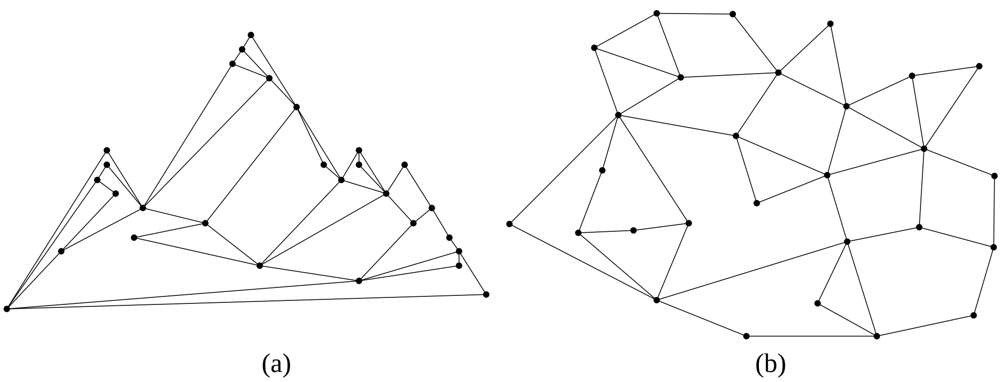


Figure 2: The graph from Fig. 1 as drawn by our system.

w in Fig. 1, which was drawn based only on dummy edges that were removed in the final drawing. Fig. 2(a) shows the same graph, drawn using our variant of the algorithm, and Fig. 2(b) shows the final result. Another example of the results of our algorithm is given in Figure 8, which contains a planar graph of 49 vertices.

As mentioned, a more detailed description of this algorithm appears in [HS93].

4 Planar graphs: Some enhancements

We have incorporated a number of heuristics and enhancements that improve the drawing algorithm of phase C. They are:

1. Choosing the initial edge for the algorithm of phase C in such a way that the external face in the drawing will be the longest face of the graph.
2. Adding dummy edges, so that the graph that is input to phase C becomes biconnected.
3. "Centralizing" each vertex with respect to its neighbors prior to the randomized algorithm of phase D.
4. Preventing the randomized algorithm of phase D from introducing new edge crossings.

We now discuss each of these briefly.

4.1 Pull a long face

The embedding list that is input to the drawing algorithm of phase C does not enforce a choice of the face to be made external. In fact, a graph can be drawn with any of its faces as external, without affecting the embedding list. Since one of the criteria for nice drawings is to have short edges, while maintaining uniform distribution of the vertices, a good heuristic would be to choose a face with a maximal number of vertices along its boundary. This gives rise to a large drawing space, enclosed by an external face with relatively short edges.

Given the embedding lists, this heuristic is easy to implement. However, it is worth mentioning that, in general, embedding lists are not unique[3], and different embeddings can give rise to longest faces of different size. A good implementation of the longest face heuristic would be to examine all possible embeddings and choose the one with the *longest* longest face. This can be done with the variant of the PQ-trees embedding algorithm that produces *all* possible embeddings, described in [CNAO85]. This should be done with care, however, as there might be exponentially many different embeddings for a given graph. An implementation of this idea can be to cut off the exhaustive search for embeddings once a predefined number of them was constructed, and then choose the best one among the available ones. We have not implemented this version, and leave it for further improvements.

4.2 Use your biconnections

Our drawing algorithm in phase C requires the input graph to be biconnected, a fact that is crucial to the existence part of its correctness proof (see [HS93]), and we have not been able to generalize it to deal directly with general graphs.

[3]In fact, a planar graph has a unique embedding only if it is triconnected [Whi32].

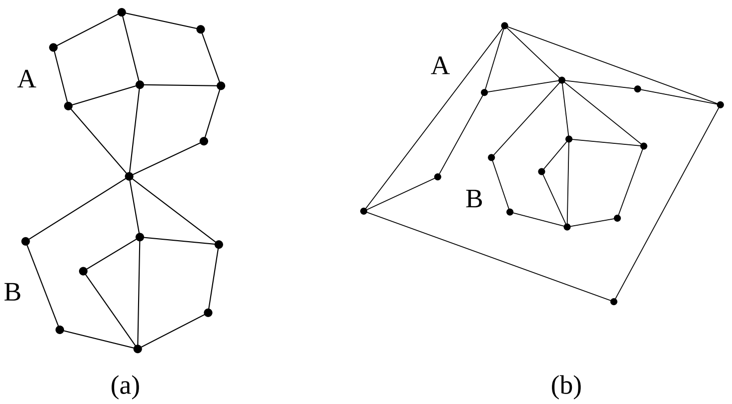


Figure 3: The unified face of two biconnected components, shown after removal of the dummy edge, and drawn as external (a), and interior (b).

Hence, for non-biconnected graphs we have incorporated the following preliminary step, that makes the graph biconnected by adding dummy edges. (These are removed prior to phase D, of course.)

Let A and B be two biconnected components of a planar graph G, that have a common vertex v (v is thus a cut-vertex, whose removal will disconnect A from B). We turn A and B into a single biconnected component by chosing two vertices $u \in A$ and $w \in B$, both neighbors of v, and adding a dummy edge between them. It is important to realize that this cannot destroy planarity. However, the particular pair chosen does affect the topological embedding of the large component: Depending on the pair of vertices chosen, the operation merges a pair of faces, one from each component, into a single face.

As before, we would like to choose u and w such that the merged face will be as long as possible. If this face turns out later to be the external one, then the two components will be drawn as adjacent portions of the graph, connected at the cut-vertex, and having a long external face (see Fig. 3(a)). Even if the merged face is not destined to become external, the result is better when this internal face is long, since one component will be contained in its entirety inside one of the faces of the other, as in Fig. 3(b); having the external face of the inner component be larger, as well as the inner containing face of the outer component, clearly yields a better spreadout of the vertices.

Tracking the longest face for each vertex as components are merged requires $O(n^2)$ running time.

4.3 Play center field

The output of the algorithm of phase C has a typical triangle-like form. The edge chosen to be initial is drawn as the basis of the triangle, and it is the longest edge in the drawing. In general, lower edges come out longer and higher ones shorter (see Fig. 2(a)). Submitting this output, as is, to the randomized phase is not very wise, since a large number of iterations are needed to overcome the variance in edge lengths.

We would like to break the typical structure of the output, by moving every vertex towards the center of gravity of its neighbors, as long as no crossings emerge.

This is achieved by progressing backwards along the straight line from the desired position to the current position, through some constant number of "stations" (7 in our implementation). The process stops when no crossings are formed. Thus, the vertex is left at the station closest to the center that still preserves planarity. (Singly-neighbored vertices are placed at a predefined distance from their neighbor.) [4]

Since the centering algorithm processes the vertices one by one, the overall result can be far from optimal. After a vertex is centralized, some of its neighbors might be centralized in subsequent steps of the process, possibly leaving it far from the center of its neighbors' final positions. Hence, in terms of optimality, we cannot expect much from this part of the system. However, in practice it does a pretty good job. The typical triangular shape of the output from the drawing algorithm is broken, the drawing has a far smaller variance in edge lengths and is more appropriate as an input to the randomized phase.

As far as complexity goes, testing a new position for a vertex requires $O(ne)$ running time to re-evaluate the edge crossings component of the cost function (see [DH89]). As this is done n times here, we have a running time of $O(n^2e)$.

4.4 Do not cross

The randomized phase D is carried out as described in Section 2, with one exception. Since only an overall improvement in the value of the cost function counts, it is possible that a new position for a vertex will be accepted despite the fact that edge crossings emerge. This will happen if other components of the cost function, i.e., edges lengths and the distribution of vertices, are greatly improved, but only a small number of crossings emerge. We believe

[4] The idea of placing each node at the center of gravity of its neighbors as a criterion for aesthetics in graph drawing, is the basis of Tutte's algorithm [Tut63] for drawing triconnected planar graphs. We incorporate this idea here for non-planar graphs, in a totally different way.

that a planar graph should be drawn planar even at the cost of some distortions. Therefore, in the case of planar graphs, we have implemented an explicit rejection of moves that result in edge crossings. Thus, the graph is kept planar throughout the randomized phase, even when this entails rejecting moves that improve the overall score.

5 Non-planar graphs

Our first attempt at drawing non-planar graphs was to submit the planar subgraph found by the algorithm of phase B^-, together with the embedding found in phase B, to the drawing algorithm of phase C, and then to reinsert the edges removed, letting the randomized phase take care of beautification. This naive approach proved to be problematic. Reinserting even a small number of edges into the planar drawing created by the drawing algorithm produced situations that were very hard for the randomized phase to deal with. The number of crossings was often large, and the edges reinserted were long. The performance of the randomized phase was poor, sometimes even worse than its performance on random initial layouts of the graph. The reason is, of course, that although the planar subgraph phase attempts to minimize the number of edges removed and then reinserted, it does not do well in minimizing the number of crossings or edge lengths, which are the kinds of difficulties that can be very hard for an algorithm that moves one vertex at a time to deal with.

5.1 Add dummies smartly

To solve the aforementioned problem, we have developed a more elaborate algorithm, that reinserts the edges *before* executing the drawing algorithm, rather than after it; this is phase B^+. When reinserting an edge, we keep the graph planar by creating dummy vertices in places where crossings occur. However, we would like to reinsert an edge while introducing as few dummy vertices as possible. This is done using a "shortest path of faces" heuristic, as follows:

Assume we have a planar subgraph G_p of the original input graph G, along with a planar embedding thereof. Let $e \in G - G_p$ be one of the extracted edges, $e = (u, v)$. To insert e into G_p causing as few new crossing points as possible, we carry out a breadth-first shortest path search in the *dual* graph of G_p, i.e., the graph of its faces. (Two faces are neighbors if they have a common edge in G_p.) The search starts with all faces whose boundary contains vertex u, and terminates at the first face whose boundary contains vertex v.

After the shortest path is found, the edge e is inserted into G_p as a sequence of edges that traverses this path of faces, by introducing dummy vertices where

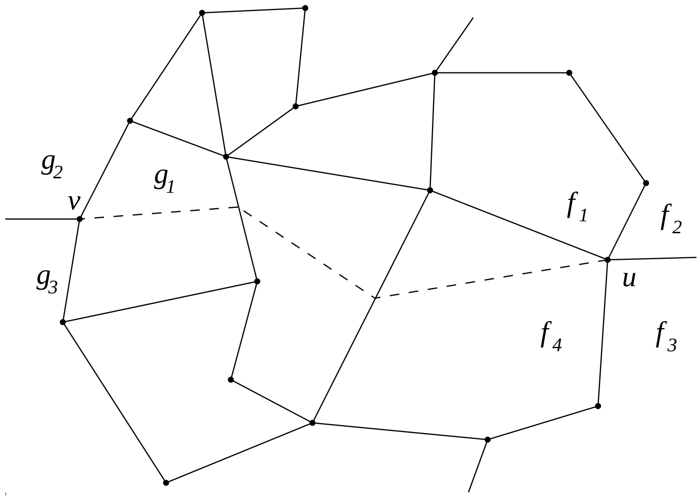


Figure 4: Adding a quasi-edge using the short face-path heuristic.

crosses occur. We call such a sequence a *quasi-edge*; see Fig. 4. This process is performed repeatedly, reinserting the extracted edges one by one, enriching (but maintaining the planarity of) the graph G_p at each step.

The final graph, call it G', is a *quasi-planarization* of the original input graph G, in the following sense. Its vertices contain the vertices of G with some additional dummy vertices, all of which occur along quasi-edges. The edges of G are mapped into edges or quasi-edges in G'.[5]

5.2 Straighten things out

The (planar) graph that results from the shortest face-path heuristic is then submitted to the drawing algorithm of phase C. A dummy vertex is not shown as a vertex, but as a pair of bends, one on the quasi-edge and one on the edge it crosses. However, since our goal is to produce a straight-line graph, we would like the randomized algorithm in phase D to try reduce the number of the bends without increasing the number of crossings by too much. For this purpose we have enriched the randomized phase in the following two ways:

First, if we can straighten such a pair of bends without causing any damage, i.e., without increasing the number of crossings (except for the single crossing that is presumably needed to replace the pair of bends itself), we do it. Testing for this is carried out in two stages, one for each of the two bent edges involved, as illustrated in Fig. 5. If both bends pass the test process, the dummy vertex

[5] As in the longest face heuristic of Section 4.1, a better implementation of this process would be to check all possible embeddings of the planar subgraph G_p, choosing the one that minimizes the number of dummy vertices. We have not implemented this.

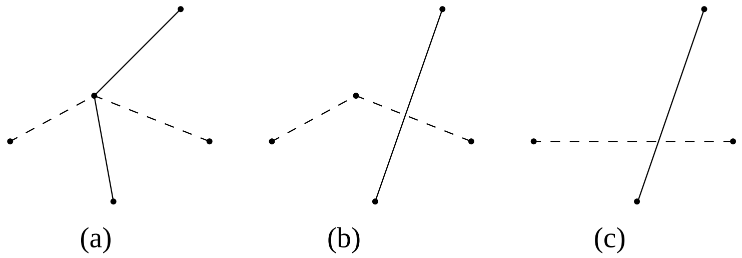


Figure 5: Straightening bends in two stages: (a) the dummy vertex, (b) after the first stage, (c) after the second stage.

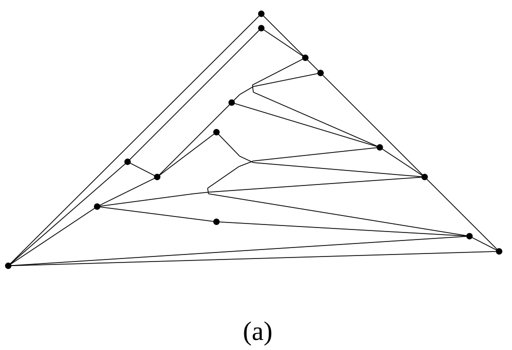


Figure 6: A graph drawn using the shortest face-path heuristic.

that caused the bends is eliminated, and the edges are straightened, as in Fig. 5(c). This replacement and elimination procedure is executed several times during the randomized phase.

Figure 6(a) shows a graph after the shortest face-path heuristic, in which 3 dummy vertices appear. Its final form is given in Fig 6(b), in which all 3 dummy vertices were eliminated, resulting in 3 crossings in the drawing.

The second modification to the randomized phase is a new component added to the cost function. It embodies a heuristic, to the effect that the chance of eliminating bends increases as the angles involved come closer to being straight. For each dummy vertex v in the graph and each of the two quasi-edges that pass through it, if the quasi-edge bends at v with angle α, the following term is added to the cost function:

$$\left(\lambda_6 \cos(\frac{\alpha}{2})\right)^2$$

This term yields small values for α close to π, and larger values for sharper angles, as needed.

5.3 Cross if convenient

Recall the strategy we adopted in the randomized phase of the planar case, whereby moves that introduce edge crossings are rejected, even if they improve the overall value of the cost function. In the non-planar case, this strategy leads to poor performance as far as the straight-angles heuristic goes, in that it tends to leave more dummy vertices intact. These vertices were usually eliminated completely when subjected to moves that improve the overall cost function, even at the expense of introducing new crossings. Moreover, the number of crossings did not rise dramatically by this. Hence, for non-planar graphs we have decided not to reject moves that increase the number of crossings.

6 Examples

Many examples are provided in [DH89]. They demonstrate the power of the simulated annealing approach for graphs of modest size or simple structure, such as the 3-dimensional cube (8 vertices), the dodecahedron (20 vertices), the 6-by-6 grid, and various trees. For such graphs, the preliminary phases of our system do not provide much added value. Hence, in this section we concentrate on cases where a significant improvement over the bare randomized phase is achieved.

6.1 Planar graphs

Planar graphs of any size are drawn planar by our system, while the annealing system of [DH89] has difficulty achieving planar drawings for some graphs of even moderate size. Fig. 7 shows the output of our system on a planar graph of 49 vertices and 112 edges. Fig. 8 shows the intermediate result, as output from phase C and prior to phase D. This demonstrates the power of the randomized algorithm of [DH89] in taking an "ugly", but planar, version of the graph and drawing it nicely. In contrast, when applied to a random layout, without the heavy-duty preprocessing of phases B and C, the system of [DH89] does quite poorly, as can be seen in Figure 9. Although some of the graphs inherent structure can be seen, the drawing is far from optimal; it seems to need some sort of turning things "inside out".

Repeated runs of the annealing system on this graph, starting from other initial random drawings, produced other results, none of them planar, and many even worse than the one shown here. This also illustrates the difference in stability between our system and that of [DH89]. When applied repeatedly to a difficult example, the latter system yields results with large variance, which is true even when it is always run on the same initial random drawing. In our

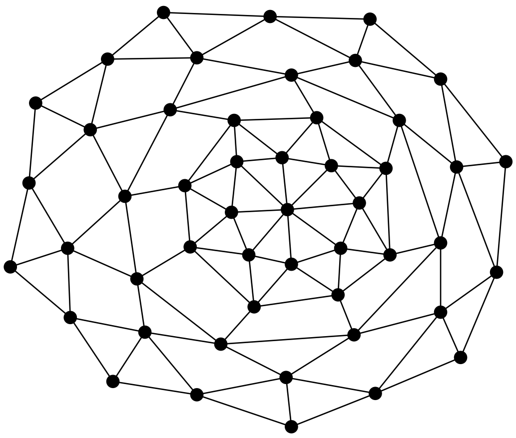


Figure 7: Final result on a planar graph of 49 vertices and 112 edges.

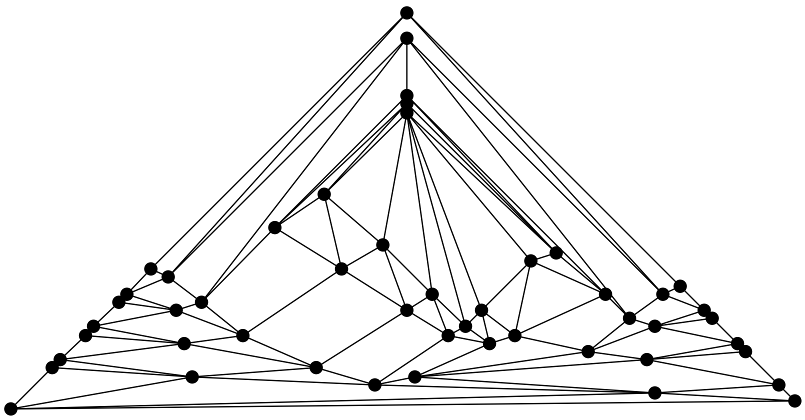


Figure 8: Intermediate result of Fig. 7 after the drawing algorithm

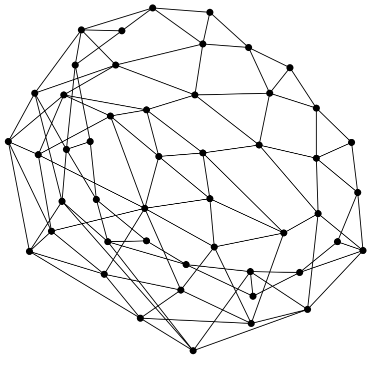


Figure 9: The graph of Fig. 7 as produced by the simulated annealing system of [DH89].

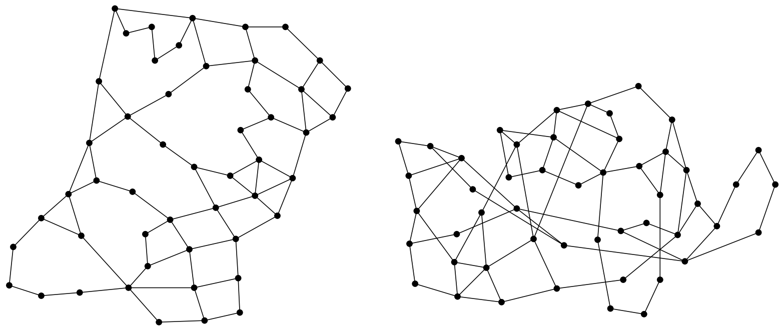


Figure 10: A sparse planar graph: (a) our system's output, and (b) that of the annealing system of [DH89].

system, on the other hand, repeated runs on the same graph yield very similar (albeit not always identical) results; this is due to the planarizing phases, and the fact that the randomized phase tends not to distroy the graph's overall topology (i.e., it's embedding).

Fig. 10 contains another example of a planar graph, this time a sparse graph with 50 vertices and 75 edges. Again, the annealing system was not able to produce a planar drawing.

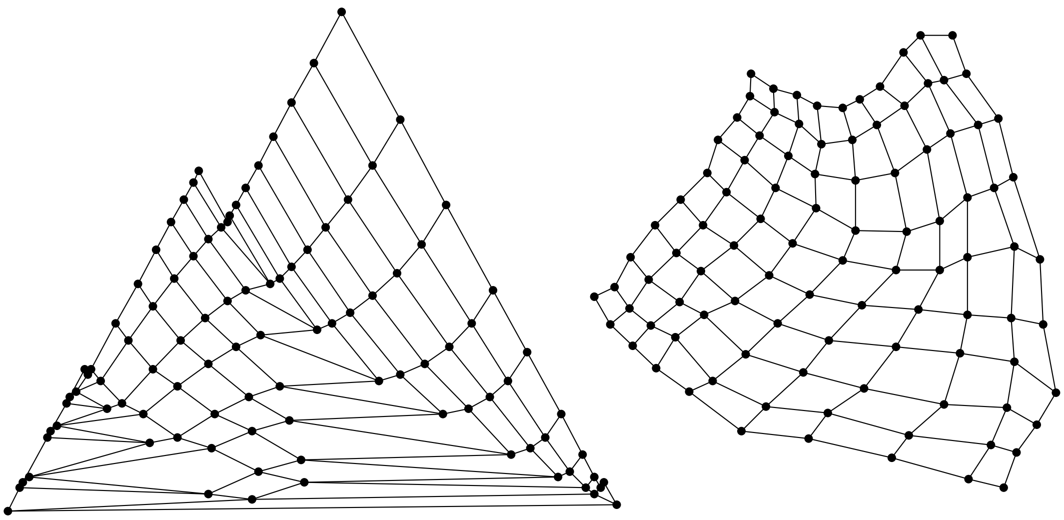
As a third example of a planar graph, consider the 100 vertex, 180 edge, 10-by-10 grid. Fig. 11(b) shows the somewhat distorted output that the system provides. The distortion originates in the planar drawing phase, whose output is shown in Fig. 11(a). The randomized phase does a lot of good, and the essential structure of this graph comes out loud and clear, yet it is unable to completely overcome the distortion.

6.2 Non-planar graphs

Our approach to non-planar graphs is clearly biased towards graphs with only a small "amount" of non-planarity, and its success is thus a function of this. The crucial parameter seems to be the number of dummy vertices that are added to the graph, as a result of the maximal planar subgraph algorithm of phase B^- and the minimal face-path heuristic of phase B^+.

Figs. 12 and 13 illustrate a successful case of a graph with 37 vertices and 76 edges. The planar subgraph algorithm removed 9 edges; reinserting them using the face-path heuristic produced 13 dummy vertices, seen as bends on the edges in Fig. 13. The final result, with only 8 crossings, appears in Fig. 12.

However, managing to add only a small number of dummy vertices is not



(a) (b)

Figure 11: A 10-by-10 grid: (a) after planarization, (b) the system's final result

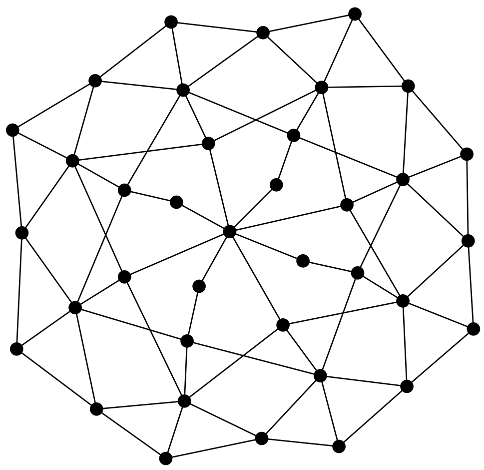


Figure 12: Final result on a non-planar graph of 37 vertices and 76 edges.

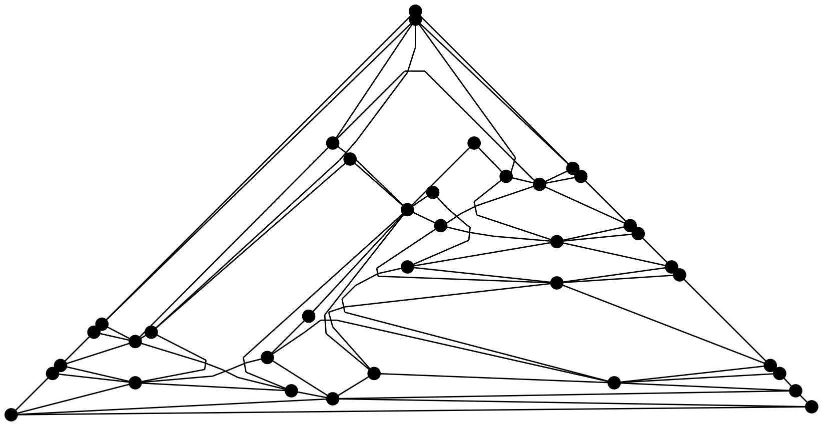


Figure 13: Intermediate result of Fig. 12 after the drawing algorithm.

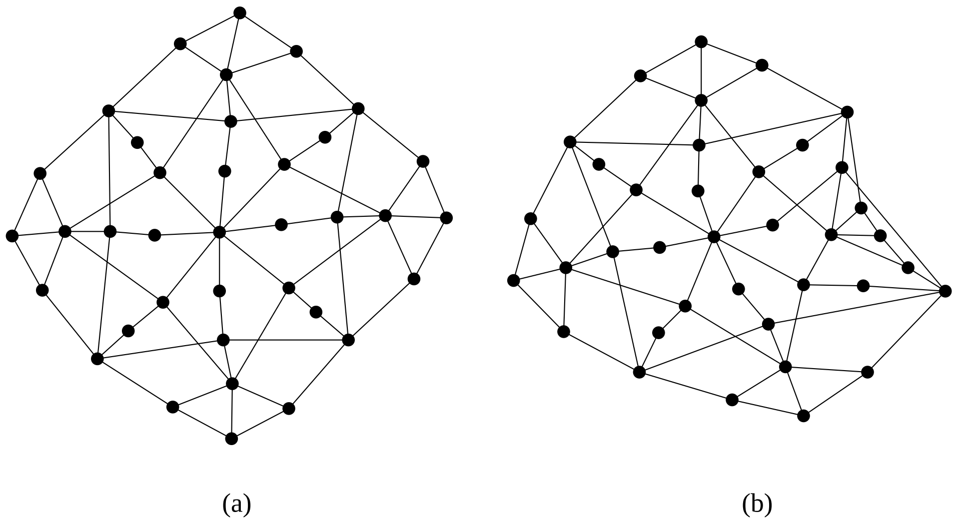


Figure 14: A less lucky example of a non-planar graph: (a) a hand-drawn version, and (b) our system's output.

always enough. Fig. 14 contains a graph with 37 vertices and 68 edges, similar to that of Fig. 12. Part (a) shows a manual drawing of the graph, in which 8 crossings occur, and part (b) shows our system's final output. The intermediate result, after phase C, yielded 11 dummy vertices by reinserting 8 edges, values that are smaller than their counterparts in the previous example. Yet the final result (Fig. 12(b)), although having only 8 crossings, just as that of Fig. 12(a), is only partially successful. The problem is due to the embedding produced in phase B, which is reflected in the topology of the final result, and upon which the randomized phase was not able to improve. It seems hard to predict such a situation in the planar embedding phase, so that this kind of phenomenon will probably have to be tolerated as is.

It is noteworthy that in both examples the number of dummy vertices produced by phase B^+ of our system is larger than the minimum possible (8 in both cases). Fortunately, the randomized phase is powerful enough to overcome redundant dummy vertices in many cases, by repeatedly effecting small changes in the drawing's topology. However, if the number of dummy vertices is significantly larger than the minimum needed, results will not be as good.

7 Complexity and performance

We first summarize the asymptotic time-complexity of our system, for an input graph $G = (V, E)$, with $n = |V|, e = |E|$. We refer to parts of the known algorithms that we use, although their details were not always discussed here, as well as to parts of our own algorithms, some of which are described more fully in [HS93].

For planar graphs, the system's phases and their running times are as follows:

Phase A:

1. Finding biconnected components — $O(n)$.
2. Calculating an st-numbering — $O(n)$.
3. Testing planarity using PQ-trees — $O(n)$.

Phase B:

1. Planar embedding of each biconnected component — $O(n)$.
2. Building faces data structure using right hand walk — $O(n^2)$.
3. Merging biconnected components to form biconnected graph — $O(n^2)$.

Phase C:

1. Planar drawing algorithm — $O(n)$.

Phase D:

1. Centering vertices — $O(n^2 e)$.
2. Randomized beautification — $O(n^2 e)$.

Since for planar graphs $e = O(n)$, the overall complexity in this case is $O(n^3)$.

For non-planar graphs, the phases and their running times are as follows[6]:

Phase A:

1. Finding biconnected components — $O(n + e)$.
2. Calculating an st-numbering — $O(n + e)$.
3. Testing planarity using PQ-trees — $O(n)$.

Phase B^-:

1. Extracting planar subgraph using PQ-trees — $O(n^2)$.
2. Making the planar subgraph maximal — $O(n^3)$.

Phase B:

1. Planar embedding of each biconnected component — $O(n)$.
2. Building faces data structure using right hand walk — $O(n^2)$.
3. Merging biconnected components to form biconnected graph — $O(n^2)$.

Phase B^+:

1. Reinserting extracted edges using the face-path heuristic — $O(e^3)$.

Phase C:

1. Planar drawing algorithm — $O(e^2)$.

Phase D:

[6] The graph $G' = (V', E')$ obtained by phase B^+ is of size $n' = O(e^2)$ and $e' = O(e^2)$). Hence, the bounds in phases C and D, which are both applied to this graph.

1. Centering vertices — $O(e^4)$.
2. Randomized beautification — $O(e^4)$.

This gives a total upper bound of $O(e^4)$ for non-planar graphs.

As can be seen, for both kinds of graphs the highest asymptotic complexity is incurred by the randomized annealing-like phase. In actual tests this phase was indeed the most costly, as the following table shows. It gives the running time (in seconds) of our system on a Sun Sparc-2 for the examples discussed in Section 6, and for some other examples, not shown in the paper. The "preprocessing" column gives the running time used by phases $A - C$ of the algorithm. The "randomized" column gives the running time of phase D, including the vertex centering step. Interestingly, the heavy-duty preprocessing that our system carries out, requires only a very small fraction of the entire running time (up to 3%).

Figure	vertices	edges	planar	preprocessing	randomized	total
7	49	112	yes	0.2	79.0	79.2
10	50	75	yes	0.2	44.1	44.3
11	100	180	yes	0.4	139.0	139.4
12	37	76	no	1.2	39.1	40.3
14	37	68	no	0.7	34.1	34.8
	64	123	no	1.3	68.3	69.6
(random)	60	80	no	0.6	37.3	37.9
(random)	60	120	no	3.9	102.5	106.4

8 Future work

Clearly, much remains to be done. Some topics that pertain to harder problems (e.g., richer graphical objects, such as curved-line graphs, hypergraphs [Ber73] or higraphs [Har88]) were alluded to in [DH89]. Some work on hypergraphs has already been done; see [BETT93]. However, even in the more humble realm of straight-line graphs, the present paper, although improving [DH89], leaves a lot to be desired. The main reason is that it is heavily oriented towards planar or close-to-planar graphs. Here are some specific directions where more work could probably be done.

8.1 Symmetry

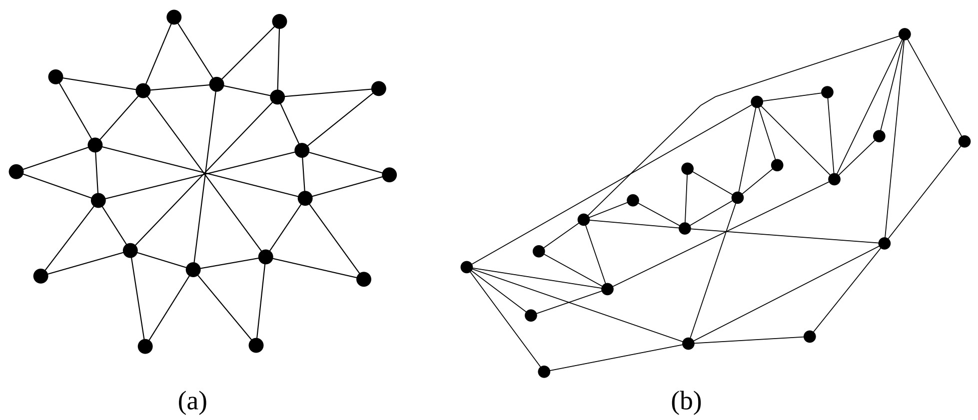


Figure 15: Symmetry vs. planarity.

The examples we have presented might give the impression that symmetry comes for free, since, although it does not look for symmetry explicitly, our system often produces drawings that are highly symmetrical. However, this is not always so, as has often been pointed out in the literature. For example, consider the graph of Fig 15(a), which was drawn by hand. It has 20 vertices and 34 edges, and in this drawing we have 5 edges, mutually inter-crossed, giving a total of 10 crossings. Running our system on it produces the far worse Fig 15.(b), in which there are only 5 crossings and one bent edge.

This graph is a hard example for the simulated annealing system of [DH89] too, as well as for other algorithms based on physical forces, such as spring-based methods (see [BETT93]). The same is true even when the weight attributed to crossings in the cost function is reduced to 0. Symmetry in such cases should be sought for explicitly, since it is hard to obtain as a by-product of other criteria for nice drawing.

As symmetry detection was recently shown to be NP-complete, [Man91], improvements based on randomization or heuristics should definitely be sought for.

8.2 Better planarization

It seems that one could develop better heuristics for planarizing non-planar graphs using a smaller number of dummy vertices (which will result in a smaller number of crossings in the final drawing). Poor performance in any of several parts of our system can be responsible for a larger-than-needed number of dummy vertices:

- The search for a maximal planar subgraph in phase B^- might cause the elimination of a larger number of edges than is really needed.

- The construction of a planar embedding in phase B, which ignores the edges to be reinserted, might produce an embedding that is problematic for the minimal face-path algorithm of phase B^+. There could exist other embeddings, in which reinserting the same set of edges produces a smaller number of crossings.

- Given an embedding and a set of edges to reinsert, the face-path algorithm might still introduce a larger number of dummy vertices than is needed, since it works sequentially, edge by edge. Reinserting the edges in a different order, or reinserting an edge via a different path of faces might decrease the overall number of dummy vertices.

These difficulties can be partly eased by incorporating randomization at points where choices are made. The user (or perhaps the system) can then carry out several runs, choosing the best. For example, there are cases where the face-path algorithm constructs two or more paths of the same length on the dual graph. The current implementation picks the first path found, while it could have made random choices among the set of possible paths. Similarly, the initial order in which the edges are reinserted could be determined randomly. Points of arbitrary choice exist in many places in the maximal planar subgraph algorithm and in the embedding algorithm, and a similar treatment can be implemented there too.

Sometimes a point of choice can be dealt with more intelligently, by taking the specific circumstances into account. For example, it might be possible to develop a new planar embedding algorithm, which at points of arbitrary choice will inspect the list of edges to be later reinserted and will choose its way accordingly.

8.3 Automatic tuning

As in the original system of [DH89], our randomized phase has various parameters, all of which have predefined values in the current implementation. Some can be adjusted by the user before running on a new graph. One of these is the very number of rounds carried out by the randomized phase. In many cases, a stable and satisfactory result is reached early in the run, and much of the costly running time of the randomized phase could be eliminated if the system were able to detect these cases and terminate without wasting time on rounds that contribute nothing. This was observed in [DH89].

The relative weights of the different components in the cost function can also be changed by the user prior to a new run. It would be nice to incorporate intelligent heuristics that would enable the system to change these in accordance with the input graph, or even during the run itself. We have made a humble step in this direction, concerning a problem we ran into with the size of the drawing: Large graphs tended to spread widely, and vertices were "pressed" against the borderline of the drawing space, due to the relative weight of the vertex-distribution component. The solution we implemented was to let the weight of the edge attraction component be dependent (in a linear fashion) on the size of the graph. Thus, for large graphs we have strong attraction forces along edges, obtaining a balance with respect to the repelling forces between vertices. This yields reasonable results, and large graphs are now drawn better.

One can think of other parameters to be adjusted automatically. For example, if the variance of edge lengths grows too large during the run, it might be beneficial to increase the weight of this component for a few rounds. We have not implemented this.

Acknowledgements: We wish to thank Goos Kant from The University of Utrecht and Matthias Stallmann from North Carolina State University for their implementations of the planar subgraph algorithm and the planar embedding algorithm, respectively. Both programs were integrated into our system with permission of their authors and saved a large amount of work. We would like to thank the two referees for their thorough reading and helpful comments, and for pointing out a number of additional references.

References

[BETT93] Di Battista, G., P. Eades, R. Tamassia and I. G. Tollis, "Algorithms for Automatic Graph Drawing: An Annotated Bibliography", Technical Report, Dept. of Computer Science, Brown University, Providence, 1993.

[Ber73] Berge, C., *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.

[BL76] Booth, K.S. and G.S. Lueker, "Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-tree Algorithms", *J. Comput. Syst. Sci.* **13** (1976), 335–379.

[CNAO85] Chiba, N., T. Nishizeki, S. Abe, and T. Ozawa, "A Linear Algorithm for Embedding Planar Graphs Using PQ-trees", *J. Comput. Syst. Sci.* **30**:1 (1985), 54–76.

[CON85] Chiba, N., K. Onoguchi, and T. Nishizeki, "Drawing Plane Graphs Nicely", *Acta Informatica*, 22:187–201, 1985.

[CP90] Chrobak, M. and T.H. Payne, "A Linear Time Algorithm for Drawing a Planar Graph on a Grid", Technical Report UCR-CSS-90-2, Dept. of Math. and Comp. Science, University of California, Riverside, CA, 1990.

[DH89] Davidson, R. and D. Harel, "Drawing Graphs Nicely Using Simulated Annealing", *Comm. Assoc. Comput. Mach.*, in press. (Also, technical report, The Weizmann Institute of Science, Rehovot, Israel, 1989; revised 1992, 1993.)

[Ead84] Eades, P., "A Heuristic for Graph Drawing", *Cong. Numer* **42** (1984), 149–160.

[FPP88] Fraysseix, H. de, J. Pach, and R. Pollack, "Small Sets Supporting Fáry Embeddings of Planar Graphs", In *Proc. 20th ACM Symp. on Theory of Comput.*, pp. 426–433, 1988.

[FR91] Fruchterman, T. and E. Reingold, "Graph Drawing by Force-Directed Placement", *Software — Practice and Experience*, 21(11):1129–1164, November 1991.

[Har88] Harel, D., "On Visual Formalisms", *Comm. Assoc. Comput. Mach.* **31** (1988), 514–530.

[HS93] Harel, D. and M. Sardas, "An Incremental Drawing Algorithm for Planar Graphs", manuscript, March 1995; submitted for publication.

[Him89] Himsolt, M., "GraphEd: An Interactive Graph Editor", In *Proc. STACS 89, Lecture notes in Computer Science*, vol. 349, pp. 532–533, Springer-Verlag, 1989.

[Him93] Himsolt, M., "A View to Graph Drawing Algorithms through GraphEd", In *Proc. of Alcom Workshop on Graph Drawing*, Paris 1993.

[JTS89] Jayakumar, R., K. Thulasiraman and M.N.S. Swamy, "$O(n^2)$ Algorithm for Graph Planarization", *IEEE Trans. on Computer-Aided Design* **8**:3 (1989), 257–267.

[J+91] Jones, S., A. Moran, N. Ward, G. Delott and R. Tamassia, "A Note on Planar Graph Drawing Algorithms", Technical report 216, Dept. of Computer Science, Univ. of Queensland, Australia, 1991.

[KK89] Kamada, T., and S. Kawai, "An Algorithm for Drawing General Undirected Graphs", *Inf. Proc. Lett.* **31** (1989), 7–15.

[Kan92] Kant, G., "An $O(n^2)$ Maximal Planarization Algorithm Based on PQ-trees", Technical Report RUU-CS-92-03, Dept. of Computer Science, Utrecht University, The Netherlands, 1992.

[LA87] Laarhoven, P.J.M. van and E.H.L. Aarts, *Simulated Annealing: Theory and Applications*, D. Reidel Publishing Co., Dortrecht, 1987.

[LEC67] Lempel, A., S. Even and I. Cederbaum, "An Algorithm for Planarity Testing of Graphs", In *Theory of Graphs: International Symposium* (P. Rosenstiehl, Ed.), Gordon and Breach, New York, 1967, pp. 215–232.

[Man91] Manning, J., "Computational Complexity of Geometric Symmetry Detection in Graphs", *Lecture Notes in Computer Science*, Vol. 507 pp 1–7 Springer-Verlag, Jun 1991.

[Rea87] Read, R.C., "A New Method for Drawing a Planar Graph Given the Order of Edges at Each Vertex", *Cong. Numer.* **56** (1987), 31–44.

[Sar93] Sardas, M., "Drawing Graphs Nicely on the Plane", M.Sc. Thesis, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel, 1993.

[Sch90] Schnyder, W., "Embedding Planar Graphs on the Grid", In *Proc. of the 1st Anual ACM-SIAM Symposium on Discrete Algorithms*, pp. 138–148, ACM Press, New York, 1990.

[TBB88] Tamassia, R., G. Di Battista and C. Batini, "Automatic Graph Drawing and Readability of Diagrams", *IEEE Trans. on Systems, Man and Cybernetics*, **18** (1988), 61–79.

[Tun92] Tunkelang, D., "An Aesthetic Layout Algorithm for Undirected Graphs", Master Thesis, M.I.T., 1992.

[Tut63] Tutte, W.T., "How to Draw a Graph", *Proc. London Math. Soc.*, Series 3, no. 13 (1963), 743–768.

[Whi32] Whitney, H., "Non-separable and Planar Graphs", *Trans. Amer. Math Soc*, 34:339–362, 1932